How LSTM Encodes Syntax: Exploring Context Vectors and Semi-Quantization on Natural Text

Chihiro Shibata
Tokyo University of Technology
shibatachh@stf.teu.ac.jp

Kei Uchiumi
Denso IT Laboratory
kuchiumi@d-itlab.co.jp

Daichi Mochihashi
The Institute of Statistical Mathematics
daichi@ism.ac.jp

Abstract

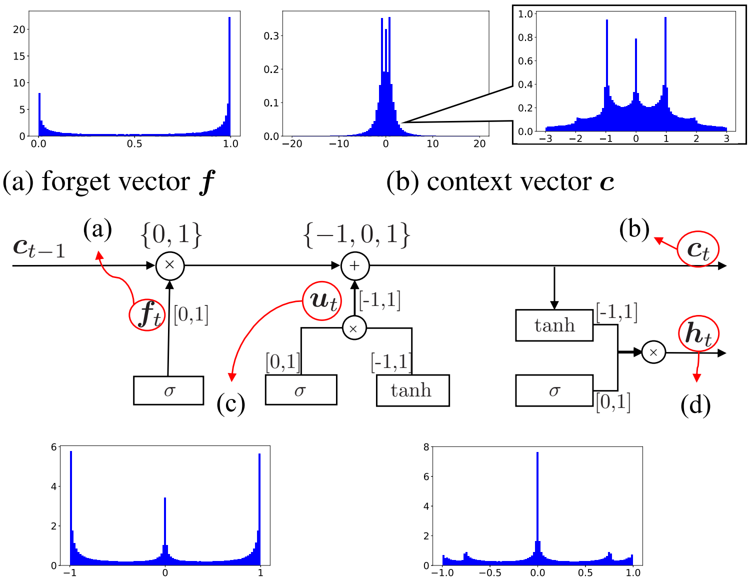
Long Short-Term Memory recurrent neural network (LSTM) is widely used and known to capture informative long-term syntactic dependencies. However, how such information are reflected in its internal vectors for natural text has not yet been sufficiently investigated. We analyze them by learning a language model where syntactic structures are implicitly given. We empirically show that the context update vectors, *i.e.* outputs of internal gates, are approximately quantized to binary or ternary values to help the language model to count the depth of nesting accurately, as Suzgun et al. (2019) recently showed for synthetic Dyck languages. For some dimensions in the context vector, we show that their activations are highly correlated with the depth of phrase structures, such as VP and NP. Moreover, with an L_1 regularization, we also found that it can be accurately predicted whether a word is inside a phrase structure or not from a small number of components of the context vector. Even for the case of learning from raw text, context vectors are still shown to correlate well with the phrase structures. Finally, we show that natural clusters of the functional words and the parts of speech that trigger phrases are represented in a small but principal subspace of the context-update vector of LSTM.

1 Introduction

LSTM (Hochreiter and Schmidhuber, 1997) is one of the most fundamental architectures that support recent developments of natural language processing. It is widely used for building accurate language models by controlling the flow of gradients and tracking informative long-distance dependencies in various tasks such as machine translation, summarization and text generation (Wu et al., 2016; See et al., 2017; Fukui et al., 2016). While attention-based models such as Transformer (Vaswani et al., 2017) and BERT (Devlin et al., 2019) and their extensions are known to encode syntactic information (Clark et al., 2019), some studies show that LSTMs are still theoretically superior in terms of ability to capture syntactic dependency (Hahn, 2019; Dai et al., 2019). Tang et al. (2018) and Mahalunkar and Kelleher (2019) also empirically demonstrate that Transformers do not outperform LSTM with respect to tasks to capture syntactic information.

Recent empirical studies attempt to explain deep neural network models and to answer the questions such as how RNNs capture the long-distance dependencies, and how abstract or syntactic information is embedded inside deep neural network models (Kuncoro et al., 2018; Karpathy et al., 2016; Blevins et al., 2018). They mainly discuss the extent to which the RNN acquires syntax by comparing experimental accuracy on some syntactic structures, such as number agreements (see Section 7 for details). Some studies also investigate in which vector spaces and layers a specific syntactic information is captured (Liu et al., 2018; Liu et al., 2019). Lately, Suzgun et al. (2019) trained LSTM on Dyck-{1,2} formal languages, and showed that it can emulate counter machines. However, no studies have shed light on the inherent mechanisms of LSTM and their relevance to its internal representation in actual text.

Weiss et al. (2018b) theoretically showed that under a realistic condition, the computational power of RNNs are much more limited than previously envisaged, despite of the fact that RNNs are Turing



(a) forget vector $\boldsymbol{f}$ (b) context vector $\boldsymbol{c}$

(c) context-update vector $\boldsymbol{u}$ (d) output-vector $\boldsymbol{h}$

Figure 1: Structure of LSTM with distributions of elements of the context vector and the surrounding internal vectors. (a) elements of forget vector $\boldsymbol{f}$ are nearly binarized to $\{0, 1\}$. (b) context-update vector $\boldsymbol{u}$ is ternarized to $\{-1, 0, 1\}$. (c) context vector $\boldsymbol{c}$ has several peaks on integers. (d) output vector $\boldsymbol{h}$ has peaks at around $\{0, \pm 0.75, \pm 1\}$.

complete (Chen et al., 2018). On the other hand, they also showed that LSTM is stronger than the simple RNN (SRNN) and GRU owing to the counting mechanism LSTM is argued to possess. Following these results, Merrill (2019) introduces an inverse temperature θ into the sigmoid and $\tanh$ functions and taking limits as $\theta \to \infty$, and thus assumes that all gates of LSTM are asymptotically quantized: e.g. $\lim_{\theta\to\infty} \sigma(\theta x) \in \{0, 1\}$ and $\lim_{\theta\to\infty} \sigma(\theta x) \tanh(\theta y) \in \{-1, 0, 1\}$. Under the above assumption, it shows LSTMs work like counter machines, or more precisely, the expressiveness of LSTMs is asymptotically equivalent to that of some subclass of counter machines. While those results are significant and giving us theoretical clues to understand how LSTMs acquire syntactic representations as their hidden vectors, it is not yet known whether or not similar phenomena occur in models learned from real-world data. Regarding this point, we show that those quantization actually often happens in real situations and bridge a gap between theories and practical models through statistical analysis of internal vectors of LSTM that are trained from both raw texts and texts augmented by implicit syntactical symbols.

We first explore the behaviors of LSTM language models (LSTM-LMs) and the representation of the syntactic structures by giving linearized syntax trees implicitly. Then, we show that LSTM also acquires a representation of syntactic information in their internal vectors even from a raw text, by statistically analyzing the internal vectors corresponding to syntactic functions. We empirically show that the representations of parts of speech such as NP and VP and syntactic functions that specific words have, both of which often act as syntactic triggers, are acquired in the space of context-update vectors, as well as syntactic dependencies are accumulated in the space of context vectors.

2 Statistics of Internal Vectors of LSTM

2.1 LSTM Language Model

In this study, we consider language models based on one-layer LSTM because our aim is to clarify how LSTM captures syntactic structures. For a sentence $w_1 w_2 \cdots w_n$, as shown in Figure 1, let $\boldsymbol{h}_t$ denote the *output vector* of an LSTM after feeding the t-th word w_t, $\boldsymbol{c}_t$ denote the *context vector*, and $\overrightarrow{w_t}$ denote the embedding of the word w_t. Let $\mathrm{LSTM}(\boldsymbol{c}, \boldsymbol{h}, \overrightarrow{w}, \Theta)$ be a function of $\boldsymbol{c}$, $\boldsymbol{h}$ and $\overrightarrow{w}$ to determine the next output and context vectors:

$$(\boldsymbol{c}_t, \boldsymbol{h}_t) = \mathrm{LSTM}\left(\boldsymbol{c}_{t-1}, \boldsymbol{h}_{t-1}, \overrightarrow{w_t}, \Theta\right), \quad (1)$$

where Θ represents the set of parameters to be optimized. The language model maximizes the probability of the next word w_{t+1} given the word sequence up to t, $w_{1:t}$:

$$p(w_{t+1}|w_{1:t}) = p(w_{t+1}|w_t, \boldsymbol{c}_t, \boldsymbol{h}_t) = s(W\boldsymbol{h}_{t+1} + b). \tag{2}$$

$s()$ is the softmax function, and W and b are a weight matrix and a bias vector, respectively. As shown in the equation (2), the history of words up to $t-1$ does not appear explicitly in the conditional part of the probability. The contextual information is represented in some form in the context vector $\boldsymbol{c}_t$ and the output vector $\boldsymbol{h}_t$. The following standard version is used as the target LSTM architecture among multiple variations (Greff et al., 2017):

$$\boldsymbol{f}_t = \sigma(A\boldsymbol{x}_t) \tag{3}$$

$$\boldsymbol{u}_t = \sigma(B\boldsymbol{x}_t) \odot \tanh(C\boldsymbol{x}_t) \tag{4}$$

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{u}_t \tag{5}$$

$$\boldsymbol{h}_t = \tanh(\boldsymbol{c}_t) \odot \sigma(D\boldsymbol{x}_t) \tag{6}$$

Here, $\odot$ is an Hadamard (element-wise) product, and $\boldsymbol{x}_t$ is the concatenated vector of $\overrightarrow{w_t}$, $\boldsymbol{c}_{t-1}$, $\boldsymbol{h}_{t-1}$, and 1. A, B, C, D are weight matrices representing affine transformation. In this paper, $\boldsymbol{u}$ and $\boldsymbol{f}$, which are derived from $\boldsymbol{x}$ by equations (3) and (4) to directly affect $\boldsymbol{c}$, are also analyzed in addition to $\boldsymbol{c}$ and $\boldsymbol{h}$. $\boldsymbol{u}$ and $\boldsymbol{f}$ are called *context-update* vector and *forget* vector hereinafter.

2.2 Internal Vectors are Naturally Quantized

The fundamental focus of this study is a natural *semi-quantization* of $\boldsymbol{f}$, $\boldsymbol{c}$, $\boldsymbol{u}$, and $\boldsymbol{h}$, as the result of learning. First, each element of $\boldsymbol{u}$ is approximately quantized, or ternarized, to $\{-1, 0, 1\}$ as shown in Figure 1(c). This discretization is a consequence of equation (4): the distribution of the first term is almost concentrated on 0 and 1, and that of the second term is concentrated on ± 1. We experimentally confirmed that even if each element of $\boldsymbol{u}$ is strictly ternarized by thresholds, it does not lose important information. For example, Table 1 lists the most similar words with the word "her" measured by the internal vectors (see Section 6.2 for details). $\theta(\boldsymbol{u})$, which is obtained by thresholding $\boldsymbol{u}$ by ± 0.9, collects syntactically similar words as appropriately as $\boldsymbol{u}$ does.

$\boldsymbol{h}$	$\boldsymbol{c}$	$\boldsymbol{u}$	$\theta(\boldsymbol{u})$
my	his	his	his
his	mother	my	my
mother	playing	the	the
husband	mind	its	its
mind	husband	our	your
wife	matters	your	their
their	party	their	's

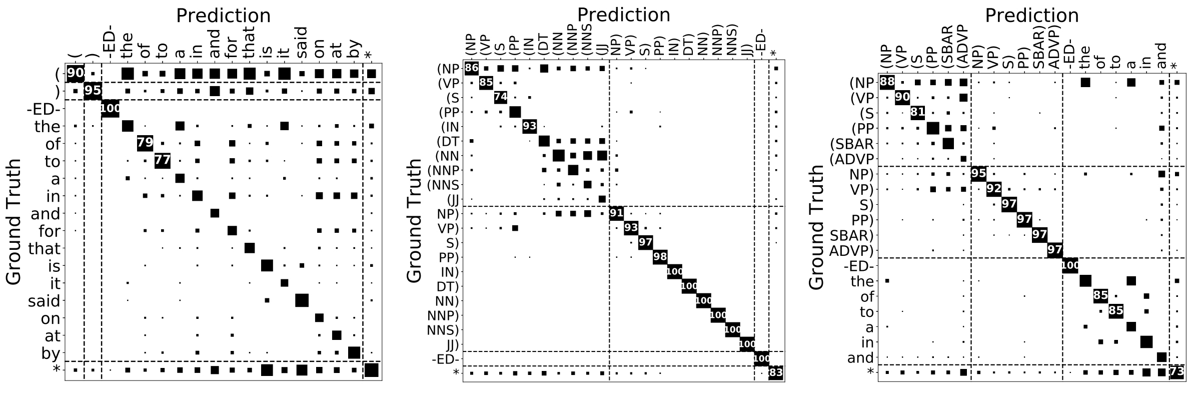
Table 1: Most similar words with "her", based on different internal vectors in LSTM. $\theta()$ is a discretization by thresholds ± 0.9.

Each element of $\boldsymbol{f}$ is also approximately binarized to $\{0, 1\}$ as seen in Figure 1(a). Context-update vector $\boldsymbol{u}$ is added to $\boldsymbol{c}$ and accumulated as long as the value of f is close to 1. Owning to the effects of such quantization and accumulation, Figure 1(b) shows that the distribution of each element of $\boldsymbol{c}$ will have peaks on integers.

As we discuss in Section 5.2, this quantization enables the accurate counting of the number of words with syntactic features such as the nesting of parenthesis. Note that Figure 1 shows the results of learning from the *raw text* of Penn Treebank WSJ corpus (Taylor et al., 2003), and the characteristics described above do not change even if the parameters such as datasets and the dimensionality of the vectors have been changed.

3 Hypotheses and Outline of Analyses

To understand the behavior of LSTM further, we try to answer two kinds of questions: (a) *what* information is relevant with the syntax, and (b) *how* this information is correlated with the syntactic behavior. In particular, we will examine: (1) which of the internal vectors (i.e. $\boldsymbol{h}$, $\boldsymbol{c}$, and $\boldsymbol{u}$) of LSTM highly correlates with the prediction of the phrase structure and its nesting (Sections 5.1 and 5.2), and (2) how well these internal vectors or some subsets of their dimensions can predict the syntactic structures (Section 5.3). Since recognition of syntax inevitably requires recognition of the part-of-speech for each word, we also investigate: (3) how the contextual part-of-speech is represented in the internal vectors of the LSTM, and how the differences between them can be captured using PCA (Section 6).



(a) Paren+W

(b) Tag

(c) Tag+W

Figure 2: Confusion matrices of next word prediction on test data. Figure (a)–(c) correspond to the datasets Paren+W, Tag, and Tag+W; dashed lines show the groups of tokens. The number within a cell shows the precision as a percentage. Only frequent words are shown and infrequent words are collectively denoted by ‘*’.

4 Target Datasets and Learned Models

4.1 Configuration of Datasets

We use sentences with syntax trees in Peen Treebank Wall Street Journal (PTB-WSJ) corpus (Marcus et al., 1994; Taylor et al., 2003) as data for training and testing. We randomly chose 10% of data for testing. Phrase structures are linearized and inserted into, or replaced with, sentences as auxiliary tokens in several manners as follows:

Paren consists of only ‘(’ and ‘)’ without words,
Paren+W consists of ‘(’ and ‘)’ and words,
Tag consists of ‘(*T*’ and ‘*T*)’ without words, where *T* represents a tag in Penn Treebank,
Tag+W consists of ‘(*T*’ and ‘*T*)’ and words,
Words is just a set of raw words.

For example, a sentence in the original data “(NP (DT a) (JJ nonexecutive) (NN director))” is converted to “(() () ())” in Paren and “(NP (DT DT) (JJ JJ) (NN NN) NP)” in Tag. The latter needs some attention; here, each space-separated token such as “(”, “(NP”, or “JJ)” is considered as a single word. The size of the vocabulary in Paren and Tag is 2 and 140, respectively. For Paren+W and Tag+W, less frequent words were replaced by their parts of speech so that the total number of words was less than 10,000. Additionally, we also included a small experiment using Lisp programs: in particular, we used `slib` standard library of `scheme` and conducted experiments under the scenarios Paren and Tag to show that LSTM also works similarly for other “languages” other than WSJ. Note that in the all scenarios above, LSTM does not know the correspondence between “(*T*” and “*T*)” for each tag *T* in advance, because these auxiliary “words” are simply converted to integers like any other words and fed to LSTM. Therefore, syntactic supervision in our experiments is not complete but only hinted.

4.2 Learning Models

The simplest architecture for LSTM language model is employed, which is composed of a single LSTM layer with a word embedding and a softmax layer. The size of the word embedding vectors and the internal vectors are determined according to the size of the vocabulary: 100 for Paren, 200 for Tag, and 1,000 for Paren+W, Tag+W, and Words because they include actual words. We used Adam (Kingma and Ba, 2015) for optimization, where hyperparameters such as the step size are the same as (Kingma and Ba, 2015). After 20 epochs of training, a model that has the best accuracy for test data among all epochs is chosen for analysis. For the dataset Words, assuming the actual usage of LSTMs, we applied dropout to the input vectors. The rates of Dropout are set to 0.2 and 0.5 for the embedding and output vectors, respectively.

4.3 Prediction Accuracies

We compare the accuracy of predicting the next word among different datasets to phenomenologically confirm the acquisition of phrase structures. As shown in Table 2, the end of sentence (EOS) is predicted by LSTM almost perfectly in terms of both precision and recall for all datasets except for Words. Because EOS occurs in a sentence if and only if the numbers of '(T' and 'T)' are equal for all T, we can conjecture that the LSTM model accurately counts the balance and the nesting of them.

Dataset	BOP	EOP	EOS	Words
Paren	0.77	0.87	**1.00**	–
Paren+W	0.90	0.96	**1.00**	0.78
Tag	0.87	0.93	**1.00**	–
Tag+W	0.89	0.96	**1.00**	0.86
Words	–	–	–	0.49

Table 2: Micro-averaged precision of prediction for the beginnings of phrases (BOPs), ends of phrases (EOPs), end of sentence (EOS), and raw words.

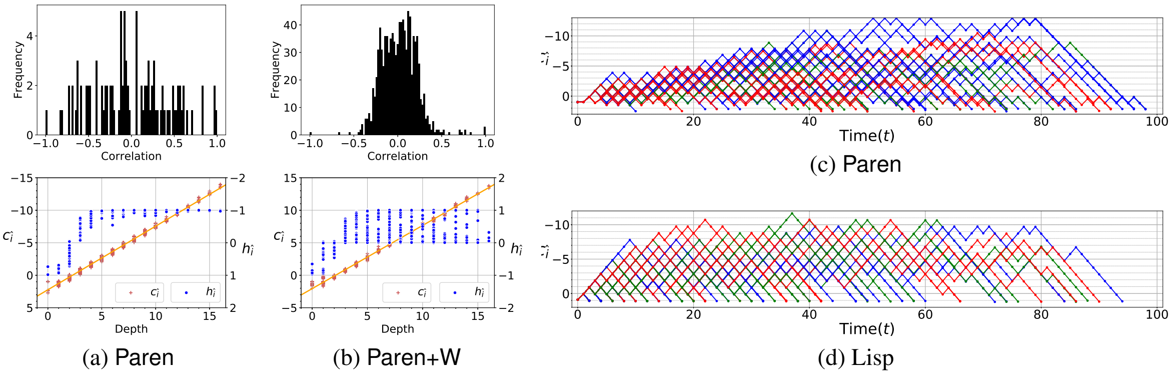
In Figure 2, the groups of Beginning of Phrase (BOP, i.e. "(T" for a tag T) and End of Phrase (EOP, i.e. "T)") are separated by the dashed lines. We can see that BOP and EOP are correctly classified across groups (Figure 2(b), 2(c)). Furthermore, each EOP is rarely misclassified to another EOP. This implies that not only the balance of the numbers of '(T'and 'T)' is completely learned, but their order of appearance is also learned quite accurately. Comparing Figure 2(c) to 2(b), we can see that the precisions for BOP and EOP are improved by including intervening words. Similarly, the precisions for the words are also improved by including BOP and EOP (Table 2). These are because the existence of words will serve as a clue to predict phrase structures, and vice versa.

5 Representation of Syntactic Structures

After these investigations on LSTM, next we will examine how each tag of the phrase structure and the depth of the nesting are embedded in its internal vectors.

5.1 Depth of Nested Phrases

We first examine the correlation coefficients between the depth of nesting and the value of each dimension of the context vector $\boldsymbol{c}$. Results are shown in the upper half of Figure 3(a) and 3(b) for Paren and Paren+W, respectively. There are some dimensions whose correlation are very high; 0.9969, 0.9978, and 0.9995 for Paren, Paren+W, and Lisp, respectively. Let $\hat{\imath}$ denote the dimension such that this correlation is maximized. As Figure 3(a) and 3(b) show, we can see that the depth of the nesting linearly correlates with $c_{\hat{\imath}}$ and almost equals to $|c_{\hat{\imath}}| - \alpha$, with some constatnt α. In contrast, the values of $h_{\hat{\imath}}$ in $\boldsymbol{h}$ are scattered; especially for Paren+W, $|h_{\hat{\imath}}|$ does not converge to 1 and has a large variance between 0



(a) Paren (b) Paren+W (d) Lisp

Figure 3: (a) and (b): (Upper) histogram of correlations with the nesting for each element of $\boldsymbol{c}$. (Lower) plots of the activations of $\boldsymbol{c}$ (red) and $\boldsymbol{h}$ (blue) for the dimension of highest correlation. (c) Value of $c_{\hat{\imath}}$ as a function of time in each sentence. Each trajectory represents a sentence in the test data. Trajectories are colored so that each one is easily distinguished. (d) same as (c) on Lisp programs. We can see that a mesh structure with the step height of approximately 1 emerges in spite of the continuous space of $\boldsymbol{c}$.

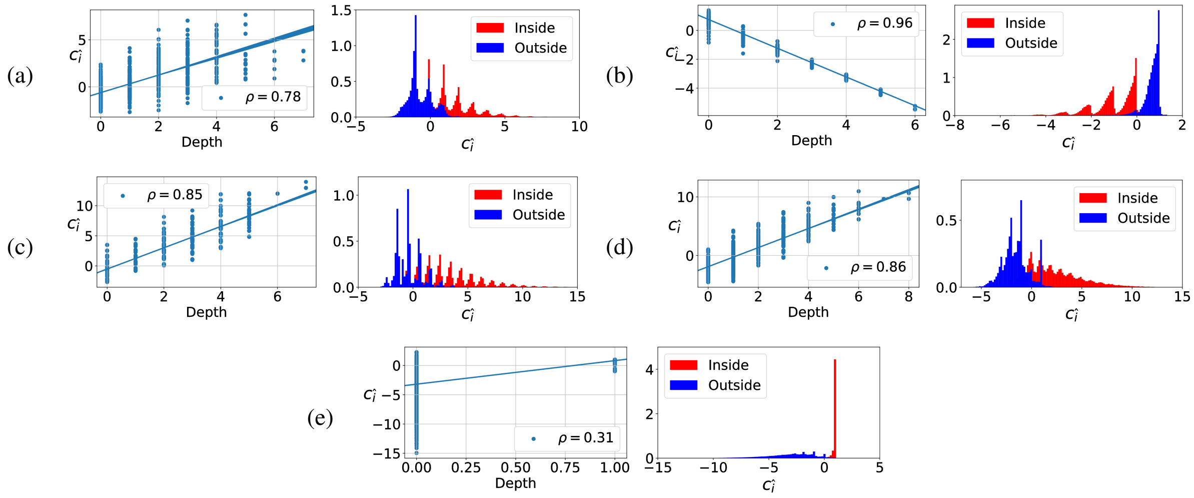


Figure 4: Relations between the depth of nesting of phrase structures and characteristic dimensions. ρ denotes the maximum correlation. The target tags are: (a)–(b) NP for Tag and Tag+W, (c)–(d) VP for Tag and Tag+W, (e) NN for Paren+W.

and 1. The first term in the right-hand side of equation (6) leads to this variance because the second term is nearly 1 or -1 when the nesting is deep.

In Figure 3(c), we randomly choose dozens of sentences from the test data whose lengths are less than 100, and plot the values of $c_{\hat{\imath}}$ as time proceeds. We can see that a mesh structure is obtained with the step height of nearly 1 in spite of the continuous space of $\boldsymbol{c}$. This is because, as described in Section 2.2, the context-update vectors $\boldsymbol{u}$ are approximately quantized so that $u_{\hat{\imath}}$ is almost binarized to ± 1. In addition, the end points of the graphs have values of approximately -2 for any sentence. This implies that the EOS can be judged easily by whether a particular dimension of $\boldsymbol{c}_t$ is approximately -2 or not.

During this study, Suzgun et al. (2019) independently discovered a similar diagram as Figure 3(c) and 3(d). However, their experiments are conducted only on a very simple formal language Dyck-{1,2} and the number of dimensions are less than 10, as opposed to our experiments in empirical data and high dimensionality of over 100 on the state vectors.

5.2 Prediction with a Single Component

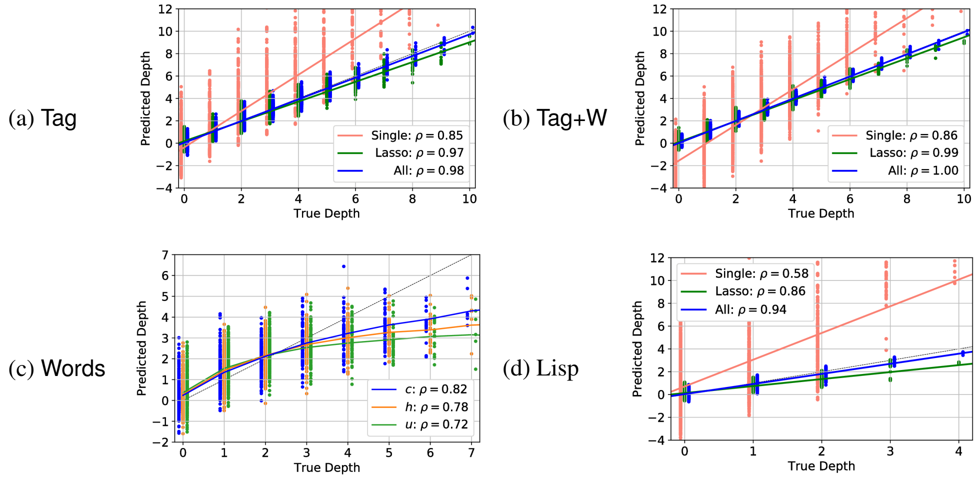
For Tag and Tag+W, there are no dimensions that completely correlate with the depth of the nesting unlike Paren and Paren+W. We extract a dimension $\hat{\imath}$ that has the largest correlation, and plot the relations between $c_{\hat{\imath}}$ and the depth of the nesting of NP and VP in Figure 4. While the absolute value of $c_{\hat{\imath}}$ increases almost linearly with the depth, its variance is not small except for NP on Tag. Thus, we cannot say that a single element of $\boldsymbol{c}$ purely encodes the depth of the nesting for a particular tag.

Dataset Tag			Dataset Tag+W			C
Acc	#nnz	ratio	Acc	#nnz	ratio	
0.996	82	41%	**0.9996**	134	13%	3×10^{-3}
0.994	56	28%	0.9992	100	10%	1×10^{-3}
0.991	34	17%	0.998	71	7%	3×10^{-4}
0.98	21	11%	0.991	51	5%	1×10^{-4}
0.96	8	4%	0.97	27	2.7%	3×10^{-5}
0.91	**5**	2.5%	0.87	**12**	1.2%	1×10^{-5}

Table 3: L_1 logistic regression from $\boldsymbol{c}$ to determine VP for Tag and Tag+W. We show the number of nonzero elements (#nnz) and its ratio for each regularization. The chance level of prediction is around 0.7.

Each of the right half of Figure 4 shows the two histograms that correspond to $c_{\hat{\imath}}$. We can observe that each activation histogram has peaks at integer values. This shows the effect of the natural quantization of $\boldsymbol{c}$. We call the ratio of the overlap of the normalized histograms as *histogram overlap ratio*. The closer the histogram overlap ratio is to 0, the higher discriminative accuracy of the dimension. The minimum histogram overlap ratio of Tag+W are 0.28 for VP and 0.06 for NN. From the perspective of histogram overlap ratio, it is easy for NN and slightly difficult for VP to classify whether a word is in that phrase by a single dimension.

For NN (common noun) tag, from Figure 4(e), it can be seen that there are no single dimension in $\boldsymbol{c}$ that highly correlate with the depth of the nesting ($\rho = 0.31$). On the other hand, the minimum histogram



(a) Tag

(b) Tag+W

(c) Words

(d) Lisp

Figure 5: Lasso (L_1) regression as a function of the nesting depth of VP for (a)-(c) and `lambda` for (d). Linear regression and $c_{\hat{i}}$ with the highest correlation coefficient are also shown. The regression results are scaled, and the depth plot is slid slightly shifted to the right for clarity.

overlap rate is 0.07, which is sufficiently low. The right histogram of the Figure 4(e) shows that the occurrence of the token '(NN' has an effect of resetting some dimension of $\boldsymbol{c}$.

5.3 Representation by a Subspace

To find a clear representation of the depth of the nesting within $\boldsymbol{c}$, we try to extract a subspace that have high correlations with it. First, we adopt a linear regression to predict the depth of nesting from $\boldsymbol{c}$. Second, we examine the number of effective dimensions; the results of regression for VP are shown in Figure 5(a)–(c). Compared with choosing the best single dimension, the correlation coefficients are clearly improved and almost equals to 1; 0.983 for Tag, 0.995 for Tag+W. This also holds for the nesting of `lambda` in Lisp programs where it is 0.940. We also empirically show that a few dimensions are sufficient to classify whether a word is in VP or not. Table 3 shows the classification accuracies: for Tag+W, we can keep the accuracy more than 0.99 while the ratio of non-zero dimensions decreases to 5%. For the case of Words, *i.e.* learning from raw text, the coefficients become smaller but still have positively correlate with $\boldsymbol{c}$, as shown in Figure 5(c); compared to $\boldsymbol{u}$, $\boldsymbol{c}$ has the smallest prediction error. In summary, the depth of the nesting of phrase structures can be represented by a sum of a relatively small number of elements of the context vector $\boldsymbol{c}$, and this relationship is approximately linear. The prediction for Words is less accurate than the other datasets with implicitly-given syntax.

6 Internal Representation of Syntactic Functions

Finally, we investigate how syntactic functions, such as part-of-speech (POS) and functional words, are represented in internal vectors when LSTM is trained for raw text. We also show that their syntactic functions are naturally represented in the context-update vector $\boldsymbol{u}$, rather than $\boldsymbol{c}$.

6.1 Representation of a Part-of-Speech

We investigate whether the LSTM-LM automatically recognizes POS when learning from raw text, because it is difficult to acquire higher phrase structures without ever recognizing POS. For this purpose, we employ a principal component analysis (PCA) to reduce the dimensionality of internal vectors of LSTM to observe unsupervised clusters. In Figure 6-(a)(b), the vertical axis denotes the standard deviation of each principal component over the observed data. The statistics over all the occurrences of words represented by the blue line shows that the variances are largely influenced by frequent words. Therefore, next we computed the principal components over unique words, as represented by the red lines. For $\boldsymbol{u}$, the standard deviations for the main components decrease after this processing. This implies that the variance within each frequent word significantly affects the result of the PCA.

“her”				“his”				“an”				“a”			
$\boldsymbol{c}$	sim.	$\boldsymbol{u}$	sim.	$\boldsymbol{c}$	sim.	$\boldsymbol{u}$	sim.	$\boldsymbol{c}$	sim.	$\boldsymbol{u}$	sim.	$\boldsymbol{c}$	sim.	$\boldsymbol{u}$	sim.
his	0.70	his	0.39	the	0.74	the	0.43	a	0.71	a	0.31	the	0.76	the	0.43
mother	0.68	my	0.33	’s	0.73	their	0.39	the	0.68	the	0.27	modest	0.76	another	0.36
playing	0.67	the	0.28	a	0.72	her	0.39	initial	0.68	its	0.26	’s	0.75	his	0.36
mind	0.66	its	0.26	their	0.71	your	0.37	enormous	0.67	another	0.25	to	0.74	your	0.34
husband	0.65	our	0.26	’	0.71	its	0.37	opportunity	0.67	her	0.25	its	0.73	’s	0.33
matters	0.65	your	0.26	her	0.70	a	0.36	planned	0.67	any	0.25	similar	0.73	every	0.33
party	0.65	their	0.25	its	0.70	’s	0.36	military	0.66	his	0.22	and	0.73	its	0.33

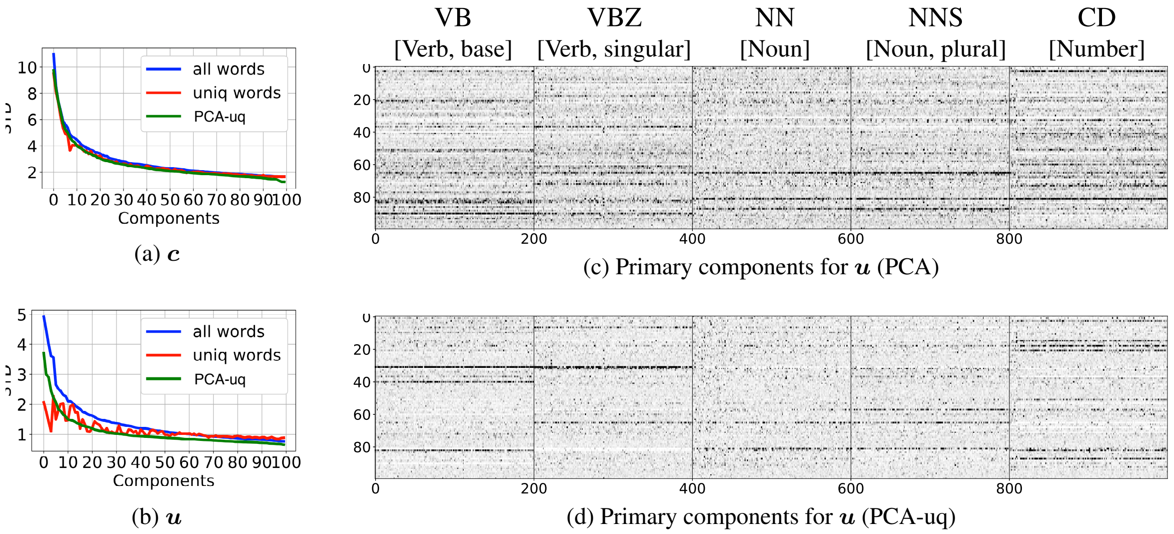
Table 4: Similarities among internal vectors for several functional words. $\boldsymbol{c}$ and $\boldsymbol{u}$ are averaged over occurrences of each word.

6.1.1 Cancelling Frequencies on PCA and Representation in $\boldsymbol{u}$

Figure 6(a) shows the top 100 components of PCA. To enhance readability of both positive and negative values, after the upper-half of the graph (x) is negatively copied to $(-x)$, each value is filtered by $\exp(\cdot)$ and shown on the y-axis. Figure 6(b) shows the effect of cancelling the frequencies of the words. In this analysis, after the number of dimensions is reduced by appling PCA to the internal vectors of all the occurrences, it is applied again to the averaged vectors, each of which corresponds to each unique word (we call this analysis as *PCA-uq*). We can see that POSs are clustered in $\boldsymbol{u}$ in an unsupervised fashion. In particular, the result of PCA-uq shows there are some dimensions that clearly distinguish similar types such as VB and VBZ, NN and NNS, and also between them. Furthermore, the distinction between verbs and nouns is evident in the first principal component of PCA-uq, at the left panel of of Figure 7.

6.2 Representation of Functional Words

Because functional words play an important role in syntactic parsing, revealing their representation in the internal vectors is important for understanding the mechanism of the syntax acquisition by LSTM. To verify if $\boldsymbol{u}$ and other internal vectors represent syntactic role of functional words, we first take the average of vectors for each word, and compute the cosine similarities between them. Table 4 lists words that have the highest similarities to some instances of words. From the tables, it can be seen that the context-update vector $\boldsymbol{u}$ captures their syntactic role more appropriately than the context vector $\boldsymbol{c}$ itself. Since $\boldsymbol{c}$ possesses contextual information in a sentence, the co-occurrence of words will affect the similarity through $\boldsymbol{c}$. We also examined $\boldsymbol{h}$ and confirmed that its clustering ability is basically similar to $\boldsymbol{c}$.



(a) $\boldsymbol{c}$

(c) Primary components for $\boldsymbol{u}$ (PCA)

(b) $\boldsymbol{u}$

(d) Primary components for $\boldsymbol{u}$ (PCA-uq)

Figure 6: (a),(b): Distribution of values for each principal component on $\boldsymbol{c}$ and $\boldsymbol{u}$. Vertical axis represents a standard deviation. (c),(d): PCA and PCA-uq (see text) results of $\boldsymbol{u}$ vectors in learned LSTM. From left, VB, VBZ, NN, NNS, and CD, respectively. Characteristic dimensions of each POS can be distinguished. Note that LSTM is learned from raw text in this scenario.

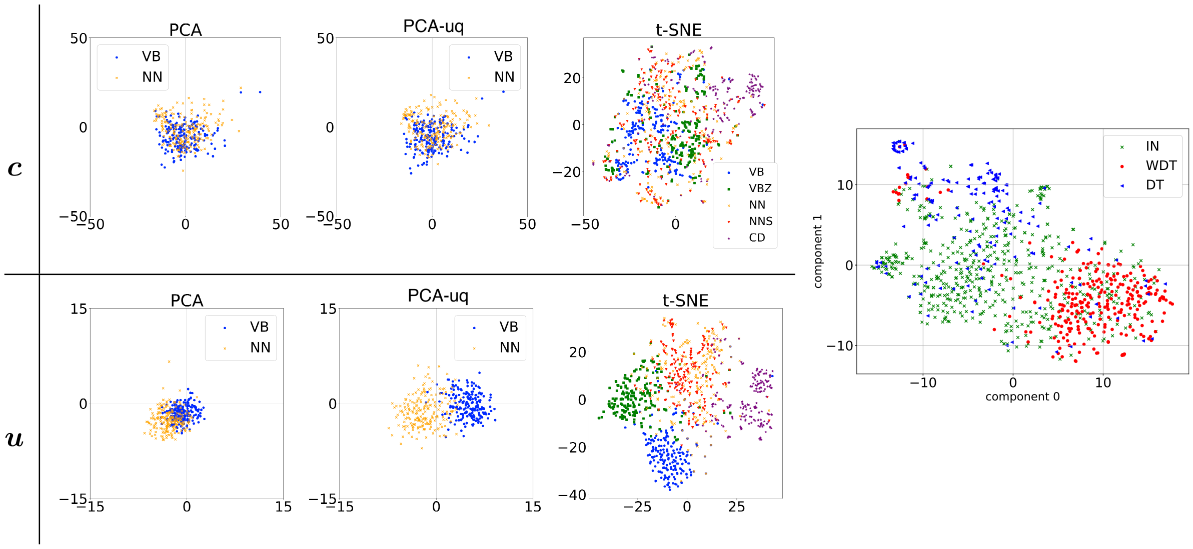


Figure 7: (Left): Result of PCA, PCA-uq, and t-SNE applied to the internal vectors: $\boldsymbol{c}$ (upper row) and $\boldsymbol{u}$ (lower row). For PCA and PCA-uq, the first and second primary components are shown. (Right): Representation of "that" in $\boldsymbol{u}$ for each usage in the corpus (t-SNE). Part of speech (not used in learning) are marked with different colors.

6.3 Representation of Ambiguity with Functional Words

A word "that" is a representative ambiguous functional word that has multiple grammatical meanings: it has three main meanings, each of which is syntactically similar to the word "*if*", "*this*", or "*which*". Figure 7 shows how these meanings are encoded in $\boldsymbol{u}$, by mapping to two dimensions using t-SNE. Although they are not completely separated, we can see that they are clustered according to their syntactic behaviors in context.

7 Related Work

As research on how LSTM tracks long-term dependence, behaviors of LSTM with several dimensions have been studied using artificial languages (Tomita, 1982; Prez-Ortiz et al., 2003; Schmidhuber, 2015). With recent applications of LSTM to various tasks, studies are being conducted on how LSTM recognizes syntax and long-term dependencies (Adi et al., 2017; Li et al., 2016). For instance, Linzen et al. (2016) uses number agreement to determine whether a language model using LSTM truly captures it. Khandelwal et al. (2018) evaluates how the distance between words affects the prediction in LSTM-LM. Weiss et al. (2018a) utilize the learned LSTM to construct deterministic automata. Furthermore, Avcu et al. (2017) control the complexity of long-range dependency using SP-k languages, and verify if LSTM can track them. Several studies have attempted to theoretically understand the learning ability of language models using RNNs, including LSTM and GRU (Cho et al., 2014; Chen et al., 2018; Weiss et al., 2018b).

8 Conclusion

In this paper, we empirically investigated various behaviors of LSTM on natural text by looking into its hidden state vectors. Contrary to previous work that deal with only artificial data, we clarified that updates $\boldsymbol{u}$ of the context vectors $\boldsymbol{c}$ are approximately discretized and accumulated in a low-dimensional subspace, leading to an approximate counter machines discussed in Section 4 and a clear representation of syntactic functions as shown in Section 5, in spite of the high dimensionality of state vectors explored in this study. Especially, we show that the representations of POS are acquired in the space of $\boldsymbol{u}$ rather than $\boldsymbol{c}$ and $\boldsymbol{h}$ in an unsupervised manner. The fact that the first principal component of PCA-uq for $\boldsymbol{u}$ encodes the difference between NP and VP is not only significant for understanding how LSTM-LM acquires ayntax, but also seen as a result of extracting the most important syntactic factor using LSTM with respect to the target language.

References

Yossi Adi, Einat Kermany, Yonatan Belikov, Ofer Lavi, and Yoav Goldberg. 2017. Fine-graind Analysis of Sentence Embeddings Using Auxiliary Prediction Tasks. In *Proceedings of the 5th International Conference on Learning Representations (ICLR 2017)*.

Enes Avcu, Chihiro Shibata, and Jeffrey Heinz. 2017. Subregular Complexity and Deep Learning. In *Proceedings of the Conference on Logic and Machine Learning in Natural Language (LaML 2017)*, pages 20–32.

Terra Blevins, Omer Levy, and Luke Zettlemoyer. 2018. Deep RNNs encode soft hierarchical syntax. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 14–19. Association for Computational Linguistics.

Yining Chen, Sorcha Gilroy, Andreas Maletti, Jonathan May, and Kevin Knight. 2018. Recurrent Neural Networks as Weighted Language Recognizers. In *Proceedings of NAACL-HLT*, pages 2261–2271.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. 2019. What Does BERT Look at? An Analysis of BERT's Attention. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 276–286. Association for Computational Linguistics.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive Language Models beyond a Fixed-Length Context. In *ACL 2019*, pages 2978–2988. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT 2019*, pages 4171–4186. Association for Computational Linguistics.

Akira Fukui, Dong Huk Park, Daylen Yang, Anna Rohrbach, Trevor Darrell, and Marcus Rohrbach. 2016. Multimodal compact bilinear pooling for visual question answering and visual grounding. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 457–468, Austin, Texas, November. Association for Computational Linguistics.

K. Greff, R. K. Srivastava, J. Koutnk, B. R. Steunebrink, and J. Schmidhuber. 2017. LSTM: A Search Space Odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232, Oct.

Michael Hahn. 2019. Theoretical Limitations of Self-Attention in Neural Sequence Models. *CoRR*, abs/1906.06755.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-term Memory. *Neural Computation*, 9:1735–1780.

Andrej Karpathy, Justin Johnson, and Li Fei Fei. 2016. Visualizing and Understanding Recurrent Networks. In *Proceedings of ICLR*, pages 1–12.

Urvashi Khandelwal, He He, Peng Qi, and Dan Jurafsky. 2018. Sharp Nearby, Fuzzy Far Away: How Neural Language Models Use Context. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 1–11.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *In Proceedings of 3rd International Conference on Learning Representations*.

Adhiguna Kuncoro, Chris Dyer, John Hale, Dani Yogatama, Stephen Clark, and Phil Blunsom. 2018. LSTMs Can Learn Syntax-Sensitive Dependencies Well, But Modeling Structure Makes Them Better. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 1426–1436. Association for Computational Linguistics.

Jiwei Li, Xinlei Chen, Eduard Hovy, and Dan Jurafsky. 2016. Visualizing and Understanding Neural Models in NLP. In *Proceedings of NAACL-HLT*, pages 681–691.

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. 2016. Assessing the Ability of LSTMs to Learn Syntax-Sensitive Dependencies. *Transactions of the Association for Computational Linguistics*, 4:521–535.

Nelson F. Liu, Omer Levy, Roy Schwartz, Chenhao Tan, and Noah A. Smith. 2018. LSTMs Exploit Linguistic Attributes of Data. In *Proceedings of The Third Workshop on Representation Learning for NLP*, pages 180–186. Association for Computational Linguistics.

Nelson F. Liu, Matt Gardner, Yonatan Belinkov, Matthew E. Peters, and Noah A. Smith. 2019. Linguistic Knowledge and Transferability of Contextual Representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1073–1094. Association for Computational Linguistics.

Abhijit Mahalunkar and John Kelleher. 2019. Multi-Element Long Distance Dependencies: Using SPk Languages to Explore the Characteristics of Long-Distance Dependencies. In *Proceedings of the Workshop on Deep Learning and Formal Languages: Building Bridges*, pages 34–43. Association for Computational Linguistics.

Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. The Penn Treebank: Annotating Predicate Argument Structure. In *Proceedings of the Workshop on Human Language Technology*, HLT '94, pages 114–119. Association for Computational Linguistics.

William Merrill. 2019. Sequential Neural Networks as Automata. In *Proceedings of the Workshop on Deep Learning and Formal Languages: Building Bridges*, pages 1–13. Association for Computational Linguistics.

Juan Antonio Prez-Ortiz, Felix A. Gers, Douglas Eck, and Jrgen Schmidhuber. 2003. Kalman filters improve LSTM network performance in problems unsolvable by traditional recurrent nets. *Neural Networks*, 16:241–250.

Jürgen Schmidhuber. 2015. Deep Learning in Neural Networks: An Overview. *Neural Networks*, 61:85–117.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks.

Mirac Suzgun, Yonatan Belinkov, Stuart Shieber, and Sebastian Gehrmann. 2019. LSTM Networks Can Perform Dynamic Counting. In *Proceedings of the Workshop on Deep Learning and Formal Languages: Building Bridges*, pages 44–54. Association for Computational Linguistics.

Gongbo Tang, Mathias Müller, Annette Rios, and Rico Sennrich. 2018. Why Self-Attention? A Targeted Evaluation of Neural Machine Translation Architectures. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4263–4272. Association for Computational Linguistics.

Ann Taylor, Mitchell Marcus, and Beatrice Santorini. 2003. *The Penn Treebank: An Overview*, volume 20 of *In Treebanks. Text, Speech and Language Technology*. Springer.

Masaru Tomita. 1982. *Learning of Construction of Finite Automata from Examples Using Hill-climbing: RR: Regular Set Recognizer*. Carnegie-Mellon University, Department of Computer Science.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, L ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Gail Weiss, Yoav Goldberg, and Eran Yahav. 2018a. Extracting Automata from Recurrent Neural Networks. In *Proceedings of the 35th International Conference on Machine Learning, PMLR*, volume 80, pages 5247–5256.

Gail Weiss, Yoav Goldberg, and Eran Yahav. 2018b. On the Practical Computational Power of Finite Precision RNNs. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 740–745.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation.